

# Automatsko objavljivanje

U okviru ove lekcije govorimo o poslednjoj fazi automatizacije procesa, objavljivanju. Govorićemo o tome šta podrazumeva ova faza razvoja i o potrebnim pripremama za testiranje programa u realnom okruženju.

Kao što smo ranije govorili, u okviru automatskog objavljivanja svaka promena bi trebalo da direktno ide u produkciju, bilo na test server ili direktno na server klijenta. U zavisnosti od izbora, može se vršiti dalje testiranje sa timom softverskih testera ili testiranje od strane korisnika. Naravno, radom u Jenkinsu smo delimično obuhvatili ovu fazu, jer smo Git repozitorijum preneli u test okruženje, gde smo vršili testove integracije.

Naravno, najbitnije je da smo Jenkins serverom postigli automatizaciju prethodnih procesa, integracije i dostavljanja, i na taj način imamo pripremljene fajlove koji se mogu dalje objaviti. Za samo objavljivanje postoji mnogo metoda, utoliko pre što nam je Jenkins omogućio sve fajlove u jednom folderu.

Kako su svi fajlovi projekta u jednom folderu, možemo npr. iskoristiti FTP klijente da izvršavamo prebacivanje fajlova iz foldera na našem računaru na folder u okviru nekog servera i to potpuno automatizovano.

Takođe, možemo iskoristiti sam Jenkins da npr. vraća fajlove na neki drugi Git repozitorijum ili da pruža fajlove na server. Problem kod ovih metoda jeste to što ili zahtevaju našu interakciju ili su rizične jer mogu izazvati greške.

Važno je skrenuti pažnju na to da je i sam proces automatskog objavljivanja ekstremno skup za kompanije, jer se korišćenjem ove metode javlja potreba za preciznim testovima u svakoj narednoj fazi projekta.

Takođe se očekuje i stabilna serverska arhitektura, jer jedan prekid u radu prekida sve faze. Stoga ovaj pristup ne možemo videti baš često u praksi, osim kod velikih kompanija i velikih projekata na kojima radi veći broj zaposlenih, gde su kompanije investirale značajne resurse u realizaciju ovih načina organizacije rada.

Ali svakako, nama je važno da se upoznamo sa procesom automatizacije, pa pogledajmo jedan od primera koji se najčešće realizuje za potrebe Pythona i automatizacije objavljivanja. Reč je o biblioteci Fabric.

## Fabric

Fabric je Python biblioteka za rukovanje udaljenim računarom pomoću SSH protokola. Pomoću Fabric biblioteke možemo koristiti komande koje nam omogućavaju kopiranje, brisanje i pokretanje fajlova na nekom udaljenom računaru ili serveru.

Ovu biblioteku možemo instalirati pomoću PIP komande sa parametrom `install fabric3`, dakle:

```
pip install fabric3
```

Ceo postupak instalacije je vrlo kratak, s obzirom na to da je ovo mala biblioteka sa jedinom svrhom da uspostavi komunikaciju između dva računara.

```
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\>pip install fabric3 1
Collecting fabric3
  Downloading Fabric3-1.14.post1-py3-none-any.whl (92 kB)
    | 92 kB 336 kB/s
Collecting paramiko<3.0,>=2.0
  Downloading paramiko-2.7.2-py2.py3-none-any.whl (206 kB)
    | 206 kB 3.3 MB/s
Requirement already satisfied: six>=1.10.0 in c:\users\dragoljub_catovic\appdata\roaming\python\python38\site-packages (from fabric3) (1.15.0)
Collecting cryptography>=2.5
  Downloading cryptography-3.4.7-cp36-abi3-win32.whl (1.4 MB)
    | 1.4 MB 1.7 MB/s
Collecting pynacl>=1.0.1
  Downloading PyNaCl-1.4.0-cp38-cp38-win32.whl (193 kB)
    | 193 kB 3.2 MB/s
Collecting bcrypt>=3.1.3
  Downloading bcrypt-3.2.0-cp36-abi3-win32.whl (27 kB)
Collecting cffi>=1.1
  Downloading cffi-1.14.5-cp38-cp38-win32.whl (167 kB)
    | 167 kB 2.2 MB/s
Collecting pycparser
  Downloading pycparser-2.20-py2.py3-none-any.whl (112 kB)
    | 112 kB 3.2 MB/s
Installing collected packages: pycparser, cffi, pynacl, cryptography, bcrypt, paramiko, fabric3
Successfully installed bcrypt-3.2.0 cffi-1.14.5 cryptography-3.4.7 fabric3-1.14.post1 paramiko-2.7.2 pycparser-2.20 pynacl-1.4.0

C:\Users\>fab 2

Fatal error: Couldn't find any fabfiles!

Remember that -f can be used to specify fabfile path, and use -h for help.

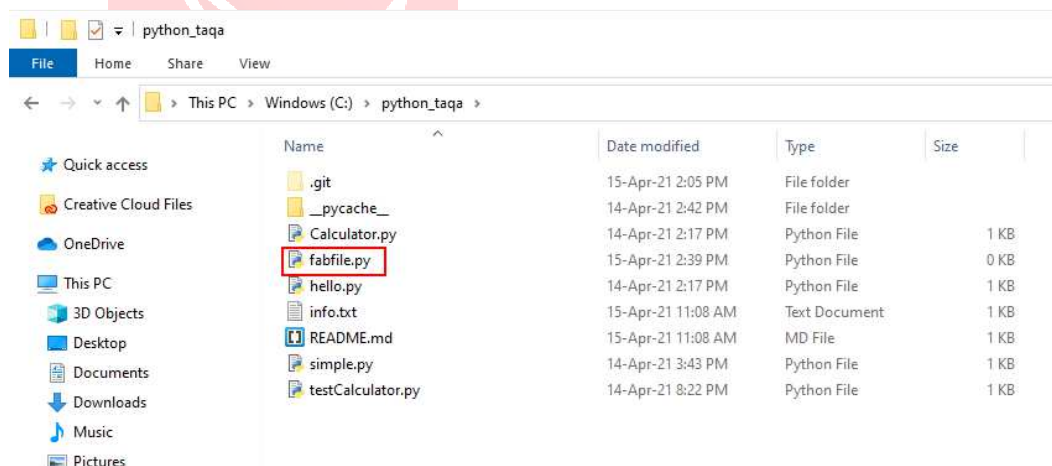
Aborting.

C:\Users\Dragoljub Catovic>
```

Slika 8.1. Pokretanje instalacije Fabric biblioteke

Kada je instalacija kompletirana, dobijamo novu skriptu i komandu koja glasi: fab. Ukoliko fab pokrenemo na samom početku, nećemo postići mnogo toga, ali dobićemo napomenu da je potrebno da kreiramo fabfile. Taj fajl, kako mu i samo ime govori, predstavlja fajl koji će biblioteka Fabric koristiti i u njemu se definišu parametri servera na koji se povezujemo, kao i to koje fajlove kopiramo.

Stoga, za naše potrebe u okviru Workspace foldera Jenkinsa – dakle, foldera gde se nalazi naš kod koji je prošao build fazu i testove integracije – definisaćemo jedan novi fajl pod nazivom: fabfile.py. U našem slučaju to izgleda ovako:



Slika 8.2. Kreiranje fabfile.py skripte

Unutar sada novokreiranog fajla, definisaćemo naše parametre. Započinjemo prvo sa importom biblioteke i njenih potrebnih paketa. Za većinu poslova sa Fabric bibliotekom dovoljne su naredne dve linije:

```
from fabric.api import env
from fabric.api import run, put
```

Sa navedenim linijama, pozivamo env module koji će nam omogućiti da definišemo server kojem pristupamo, kao i da metodom put fajl pošaljemo na server i da ga run metodom pokrenemo. Stoga, prvo ćemo definisati server na koji želimo da se povežemo i login podatke za njega:

```
env.hosts = ['example.com']
env.user = 'johnsnow'
env.password = 'developer123'
```

Dakle, ovde smo postavili podatke za adresu servera, korisničko ime i password. Naravno, ovo su samo podaci radi primera; u realnoj situaciji, podatke o serveru i korisničkom nalogu dobijate od poslodavca ili, u slučaju da ste vi zakupili hosting, od hosting kompanije.

Na kraju, potrebno je da definišemo šta Fabric biblioteka treba da uradi za nas. Kao što smo rekli na samom početku, poenta je automatizacija objavljivanja, što znači da neke od naših fajlova želimo da pošaljemo na server. Stoga možemo uraditi sledeće:

```
def deploy():
    put('simple.py', 'published/tests/simple.py')
    run('published/tests/simple.py')
```

Sada smo u okviru deploy funkcije definisali da metodom put šaljemo fajl na server. Prvi parametar metode je putanja do fajla. U našem slučaju smo već u folderu iz kojeg šaljemo podatke, pa je dovoljan naziv samog fajla. Drugi parametar predstavlja putanju gde na serveru treba postaviti fajl. U svrhe primera, naveli smo da će to biti folder published i njegov potfolder tests.

Na kraju pokrećemo simple.py fajl koji bi pokrenuo i test, što bi bio indikator uspešnog transfera. Na kraju, potrebno je samo da kroz terminal pokrenemo deploy funkciju. To postizemo komandom:

```
fab deploy
```

Dakle, koristimo komandu fab i navodimo naziv funkcije koju pokrećemo. Sa realnim podacima u našem primeru skripta bi pristupila serveru, ulogovala se u njega i prekopirala fajl sa njegovim pokretanjem. Naravno, broj fajlova za transfer se može povećavati, kao i broj pokretanja fajlova u jednom trenutku, sve zavisno od naših potreba. Dakle, ovo je u osnovi automatizacija objavljivanja, način da bez naše direktne interakcije svi fajlovi pređu na neko serversko okruženje, gde će ih dalje testirati ili QA tim ili sami korisnici.

## Dokumentovanje koda

S obzirom na to da je ova lekcija, kao i par prethodnih, posvećena automatizaciji procesa, sada ćemo govoriti o još jednom važnom aspektu razvoja softvera koji u određenoj meri možemo automatizovati, a to je dokumentovanje koda.

Dokumentacija softvera je važan prateći materijal uz svaki program. Softverska dokumentacija može biti u vidu pisanog materijala, slika, pa čak i video-instrukcija za softver koji smo izradili. Po pravilu, cilj dokumentacije je da jasno objasni kako program funkcioniše i kako se koristi. Mnogo puta do sada smo se i mi osvrnuli na dokumentaciju. Dakle, svaki put kada želimo da koristimo neku novu tehnologiju, njena dokumentacija je polazna tačka u njenom savladavanju.

Naravno, nama kao developerima u fokusu je dokumentacija koja je ugrađena u naš kod, kao i tehnička dokumentacija u kojoj objašnjavamo funkciju klase, njenih metoda, očekivane ulaze i izlaze itd.

Pored navedenog vida dokumentacije, u daljem radu uvek postoji mogućnost učestvovanja u izradi i drugih tipova dokumentacije.

Dokumentacija se u zavisnosti od ciljne grupe može podeliti na pet kategorija:

- **Specifikacija zahteva (requirement documentation)** – Ovaj vid dokumentacije služi kao podsetnik na dogovorene zahteve. U okviru nje se definišu atributi, karakteristike, prednosti i mogućnosti softvera. Pored ovih zahteva, razvojni tim na osnovu iskustva definiše i ograničenja u radu programa, zahteve kompatibilnosti, a takođe i hardverske, kao i softverske zahteve. Specifikacija zahteva se u većini slučajeva kreira na samom početku razvojnog procesa.
- **Dokumentacija arhitekture i dizajna** – Ovaj vid dokumentacije opisuje principe kojima se tim vodi tokom izrade softverskih komponenti i generalno dizajna programa. Obuhvata sve od toka podataka kroz program do arhitekture baza podataka. Često se koristi za opisivanje modula, ali samo sa aspekta – koja će biti njihova uloga i funkcija, bez detaljnog opisivanja koda unutar modula.
- **Tehnička dokumentacija** – Ovo je upravo tip dokumentacije koji smo spomenuli u uvodu u ovo poglavlje. Predstavlja dokumentaciju samog koda i u ovoj dokumentaciji će se nalaziti opisi klasa i metoda, pa sve do algoritama i generalne funkcije. Ovaj vid dokumentacije za ciljnu grupu ima druge programere.
- **Dokumentacija za krajnjeg korisnika (end user documentation)** – Često nazivana korisničkim uputstvom, predstavlja detaljne instrukcije kako koristiti program. Često pored tekstualnog materijala i primera koda sadrži slike interfejsa, pa i kratke video-instrukcije korišćenja programa. Kao što joj ime govori, namenjena je korisnicima programa, ali da napomenemo, korisnik ne mora biti samo osoba koja je kupila softver ili klijent kompanije koja je vlasnik softvera, već to mogu biti i administratori koji održavaju sistem, tehnička podrška i slično.
- **Marketinška dokumentacija** – Dokumentacija koju koriste timovi za realizaciju prodaje softvera. U dokumentaciji su predstavljene informacije koje pomažu u plasiranju proizvoda, kao i analize tržišta.

Za kreiranje svih ovih tipova dokumentacije po pravilu postoje odvojeni sektori. Tehničku dokumentaciju delimično kreiraju programeri, ali takođe i tim lideri, project menadžeri, pa i QA tim. Menadžment na projektu će u saradnji sa marketing timom kreirati promotivne materijale i dokumentaciju potrebnu za prodaju i slično.

Pogledajmo sada kako se dokumentovanje koda u osnovi odvija u okviru Pythona. Temelj svake dokumentacije biće komentar u okviru samog koda i generalno bi to izgledalo ovako:

```
class CalcClass:

    """
    Class for calculating numbers
    on init takes two integers
    and returns calculated value

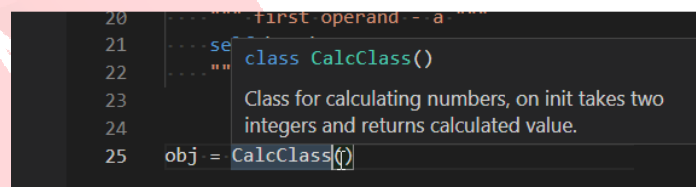
    """

    def __init__(self,a,b):
        """
        a and b operands must be passed

        Arguments:
            a: First operand
            b: Second operand
            .. todo:: Add validation"""
        self.a = a
        """ first operand - a """
        self.b = b
        """ Second operand - b"""

obj = CalcClass(2,3)
```

Kao što možete videti, sve se zasniva na upotrebi višelinijskog komentara u Pythonu. Upotreba komentara pomaže kako vama tako i kolegama iz tima u snalaženju u okviru koda. Takođe, komentar napisan unutar npr. klase pomaže i u daljem pisanju koda. Ovo se postiže u okviru naprednijih editora koji će uzeti komentar i prikazati ga kao hint kada pozovete klasu ili njenu metodu. Pa npr. u slučaju Visual Studio Code editora vidimo sledeći hint prilikom instanciranja klase:



Slika 8.3. Code hint generisan na osnovu komentara koda

Kao što možete videti, editor je iskoristio komentar unutar klase i prikazao nam ga kao pomoć. Naravno, klasu retko pozivamo u okviru fajla i komentar ovog tipa može biti ekstremno koristan kada naš program postane kompleksniji.

Mana ovog pristupa jeste što je to zamoran posao. Ovim pristupom bismo zapravo imali više linija komentara nego linija koda samog programa. Stoga se ovaj pristup ne koristi za male programe gde je lako pronaći potrebne delove, već samo kada radimo u timovima, prilikom rada na velikim projektima.

Tokom godina je došlo do potrebe da se ovaj proces u određenoj meri automatizuje, da umesto što samo vidimo komentare u okviru koda, iskoristimo te komentare da generišemo dokumentaciju u vidu html ili pdf fajla. Jedan od osnovnih programa za Python i generisanje dokumentacije je pdoc.

### Pitanje

Python biblioteka koji omogućava rukovanje udaljenim računarom nosi naziv:

- **Fabric**
- Tactic
- Static

### Objašnjenje:

*Fabric je Python biblioteka za rukovanje udaljenim računarom pomoću SSH protokola. Pomoću Fabric biblioteke možemo koristiti komande koje nam omogućavaju kopiranje, brisanje i pokretanje fajlova na nekom udaljenom računaru ili serveru.*

### pdoc

pdoc predstavlja generator dokumentacije na osnovu Python koda. Prvi je korak u automatizaciji procesa dokumentacije softvera, jednostavan je za upotrebu i vrlo brzo postiže željene rezultate.

Ovaj plugin takođe možemo instalirati putem pip-a, komandom:

```
pip install pdoc3
```

Da bismo pokrenuli proces generisanja dokumentacije, potreban nam je iskomentiran py fajl koji će plugin koristiti; uzećemo naš primer, samo malo proširen da bismo videli više detalja:

```
class CalcClass:

    """
    Class for calculating numbers,
    on init takes two integers
    and returns calculated value.

    """

    def __init__(self,a,b):
        """
        a and b operands must be passed

        Arguments:
            a: First operand
            b: Second operand
            .. todo:: Add validation
        """
        self.a = a
```

```

    """ first operand - a """
    self.b = b
    """ Second operand - b """
def add(self):
    """
    Return sum of a and b field values
    """
    return self.a + self.b

def sub(self):
    """
    Return difference of a and b field values

    Returns:
        Difference of a and b
    """
    return self.a - self.b

obj = CalcClass(2,3)

```

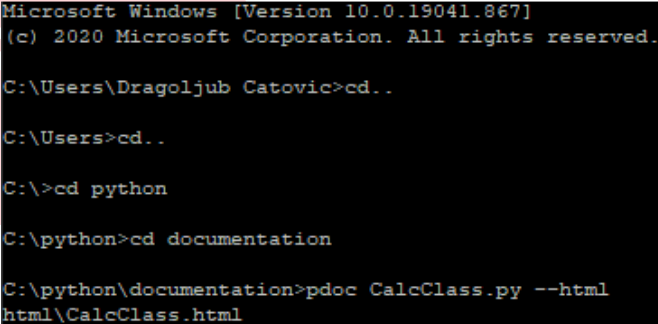
Sledeći korak je da sačuvamo ovaj fajl. Kako praksa nalaže, fajl ćemo sačuvati pod istim nazivom kao i naše klase, CalcClass.py.

Sada koristimo pdoc da na osnovu ovog fajla generiše dokumentaciju. Ovo postizemo komandom:

```
pdoc CalcClass.py --html
```

Dakle, na ovoj liniji prva stoji komanda pdoc, što aktivira plugin, sledeći parametar je tačan naziv fajla, a nakon toga -- html izdaje naredbu da se dokumentacija sačuva u .html formatu.

U konzoli (slika 8.4), možete videti da smo otišli direktno u folder gde se fajl nalazi i tu pokrenuli komandu. Ovo je nešto o čemu morate voditi računa, s obzirom na to da plugin ne može da pretražuje računar za fajlom.



```

Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Dragoljub Catovic>cd..

C:\Users>cd..

C:\>cd python

C:\python>cd documentation

C:\python\documentation>pdoc CalcClass.py --html
html\CalcClass.html

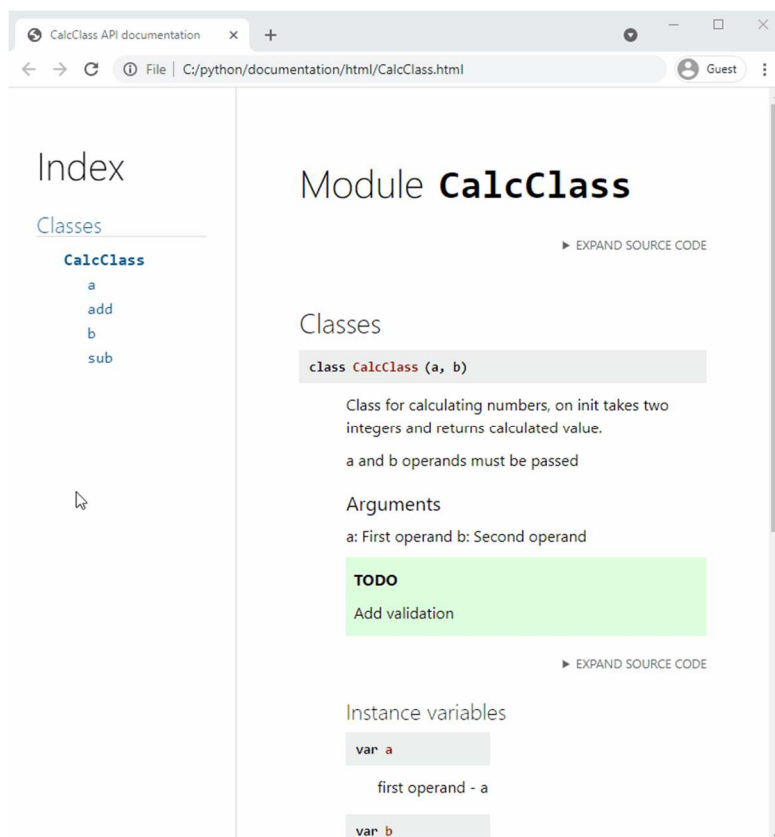
```

*Slika 8.4. Generisanje dokumentacije u vidu html fajla*

Kada je proces uspešno završen, vidimo da u narednoj liniji konzole stoji:

```
html\CalcClass.html
```

Dakle, sada na lokaciji našeg py fajla možemo pronaći novi folder pod nazivom html, u kojem se nalazi istoimeni html fajl. Kada ga pokrenemo u okviru pregledača, možemo videti sledeće:



*Animacija 8.1. Stilizovana interaktivna dokumentacija*

Kao što možete videti i na prikazanoj animaciji, sada imamo html fajl koji je potpuno generisan na osnovu naših komentara koda, sa dodatnim opcijama poput prikazivanja celog koda klase i dodatne stilizacije za neke elemente. Dakle, sa vrlo malo truda dobijamo dokumentaciju kakvu često možemo videti i u korišćenju na internetu.

Ono što je potrebno napomenuti jeste da ovaj plugin prepoznaje i par dodatnih ključnih reči u okviru komentara koda, pre svega:

```
.. todo::
```

Ova ključna reč sav tekst koji je prikazan nakon nje postavlja u zeleni pravougaonik koji možemo videti u html fajlu.

Takođe, postoje i neke dodatne komande, gde dobijamo drugu boju i oblik elementa, drugi font, dodavanje slike i slično – naravno, kada je potrebno da iskažemo važnost nekog dela koda; detaljnije o njima možete videti na sledećem linku:

<https://pdoc3.github.io/pdoc/doc/pdoc/#supported-rest-directives>



Svrha dokumentacije generisane na ovaj način jeste stvaranje mogućnosti deljenja dokumentacije koja nije nužno unutar koda. Dakle, kada je potrebno npr. da QA tim prati ispravnost dokumentacije, da tim lider prekontroliše strukturu modula i slično. Ukoliko ovaj proces ponovimo na svim modulima, vrlo lako možemo dobiti objedinjenu dokumentaciju za program, jednostavnim povezivanjem HTML fajlova. Sa druge strane, ceo proces je automatizovan, jer je nama dovoljno da unesemo jednu komandu da bismo dobili ovaj fajl.

## Rezime

- Fabric je Python biblioteka za rukovanje udaljenim računarom pomoću SSH protokola. Pomoću Fabric biblioteke možemo koristiti komande koje nam omogućavaju kopiranje, brisanje i pokretanje fajlova na nekom udaljenom računaru ili serveru.
- Fabric biblioteka se instalira komandom: `pip install fabric3`.
- Dokumentacija softvera je važan prateći materijal uz svaki program. Softverska dokumentacija može biti u vidu pisanog materijala, slika, pa čak i video-instrukcija vezanih za softver. Cilj dokumentacije je da jasno objasni kako program funkcioniše i kako se koristi.
- Dokumentacija se u zavisnosti od ciljne grupe može podeliti na pet kategorija: specifikaciju zahteva, dokumentaciju arhitekture i dizajna, tehničku dokumentaciju, end user dokumentaciju i marketinšku dokumentaciju.
- pdoc predstavlja generator dokumentacije na osnovu Python koda. Prvi je korak u automatizaciji procesa dokumentacije softvera, jednostavan je za upotrebu i vrlo brzo postiže željene rezultate.
- pdoc plugin možemo instalirati putem pipa, komandom: `pip install pdoc3`.

